Memory Management

COMP755 Advanced Operating Systems

Purpose of Memory Manager

- Find a place in RAM for programs and data.
- OS Memory Manager allocates RAM to programs and OS tasks and data.
- User level memory management is done to allocate memory for new objects.

Memory Mangement Goals

- Have the programs in memory when they need to run
- Support address relocation
- Protect one program from another
- Allow programs to share data and instructions
- Support program organization
  - read-only or execute-only segments
  - no execute stack segments
  - separate compilation

Sub Goals

- Simple to find appropriate sized space
- Simple to merge adjacent free space
- Minimize external fragmentation
- Minimize internal fragmentation
- Prevent the loss of unreturned space.
- Minimize the time required to allocate memory

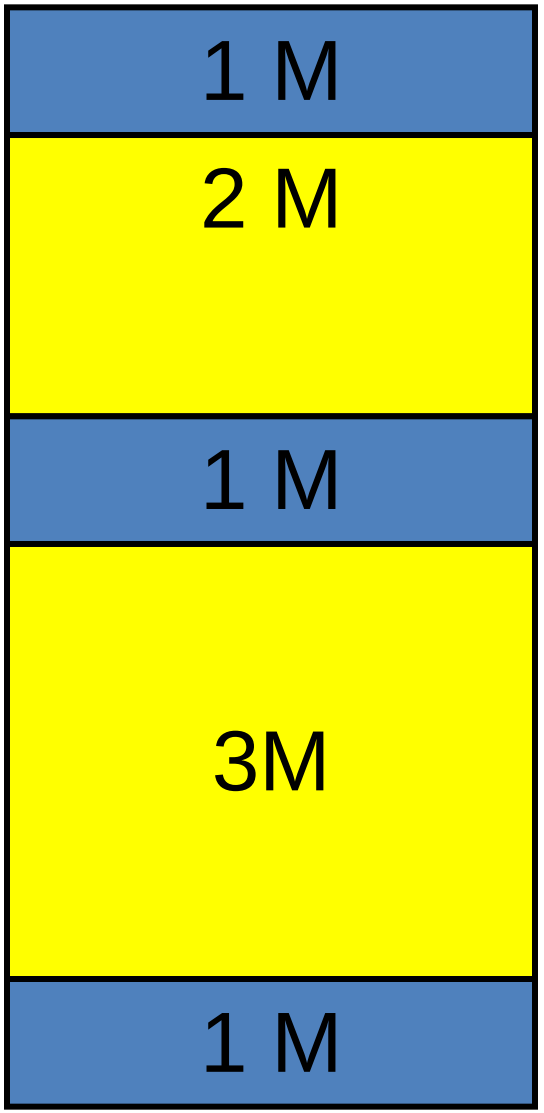
This is an example of a bin packing problem and is NP-complete even if you have all of the memory requests at one time.

An NP-Complete program is

1. non-procedural
2. $O(2^n)$
3. non-computable
4. a complete solution

Fragmented Memory

- In this memory diagram two programs use 5 MB of the 8 MB in the system.
- There is 3 MB available.
- A 2 MB program cannot be put in memory because the available space is fragmented.



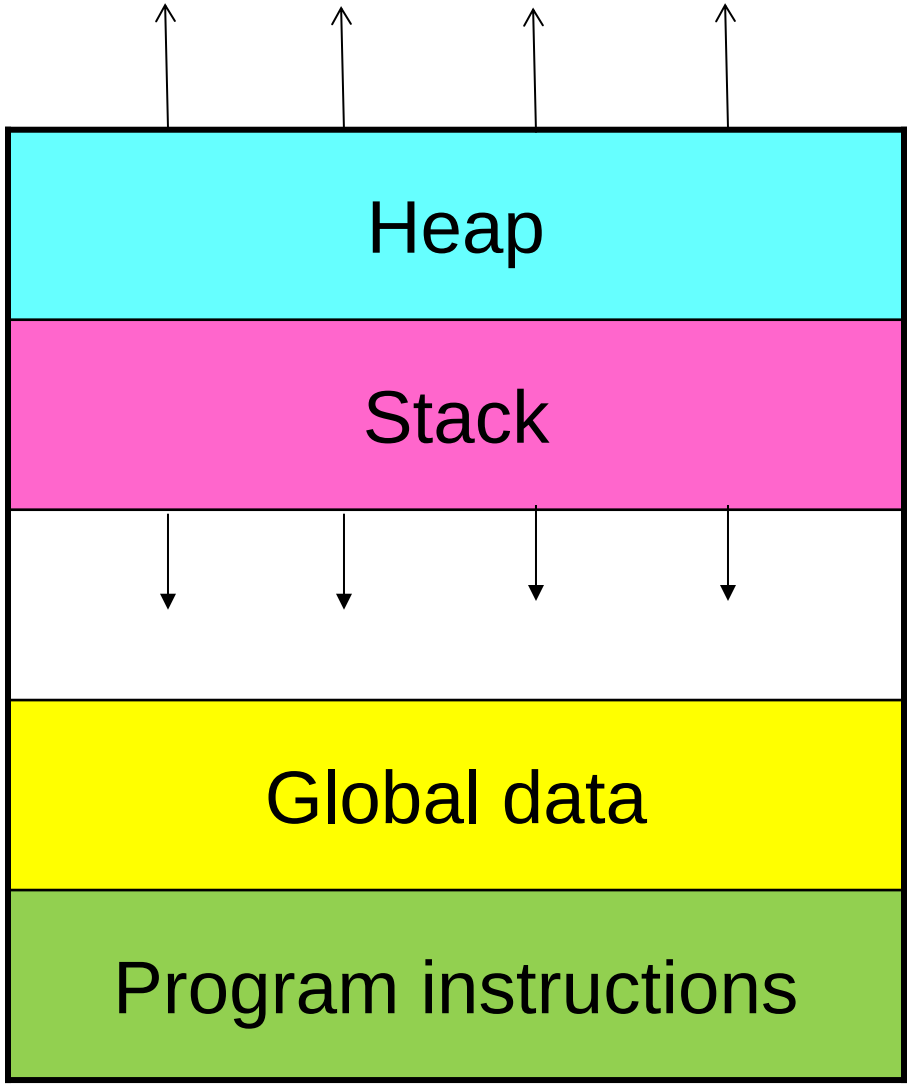
Fragmentation

- As memory is allocated and released, the usable areas become fragmented.
- **Internal fragmentation** occurs when a program is allocated to a memory area that is bigger than needed. The leftover is lost due to internal fragmentation.
- **External fragmentation** occurs when there are many small unallocated areas. These areas are hard to use because they are small and divided.

Logical Address Formation

- Compiler
  - data and instruction segments
  - addresses relative to the start of function
  - stack addresses relative to stack pointer
- Linker
  - external references resolved
  - all object file addresses are adjusted.
  - addresses relative to start of the program

Program Memory Organization



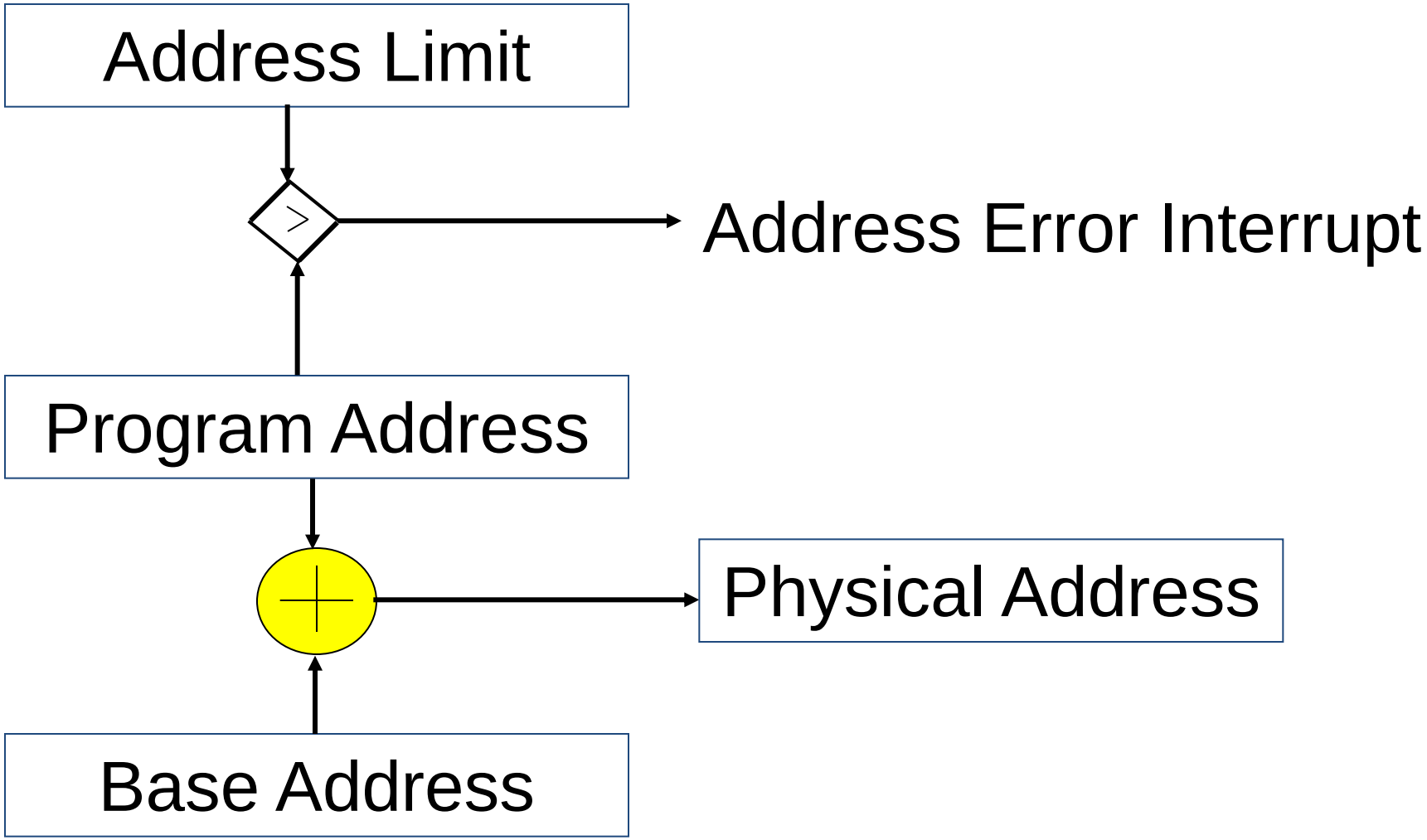
Logical to Physical Translation

- Most programs are created with zero as the lowest address in the program.
- All programs cannot be loaded at physical RAM address zero. Thus the program addresses have to be translated into hardware RAM addresses.

Simple Relocation and Protection

- For fixed and dynamic partitioning, the hardware can be designed with a base address register and a limit register.
- Every memory access is checked to ensure the address is not greater than the limit register value. If it is greater, an addressing interrupt is generated.
- The logical program address is added to base address value to get the RAM address

Address Computation



Base Address

- Programs are located in any convenient contiguous part of memory.
- When a program is to execute, the base register is loaded with the start address of the program.
- The program effective address from each memory reference is added to the value in the base register to create the physical memory location.

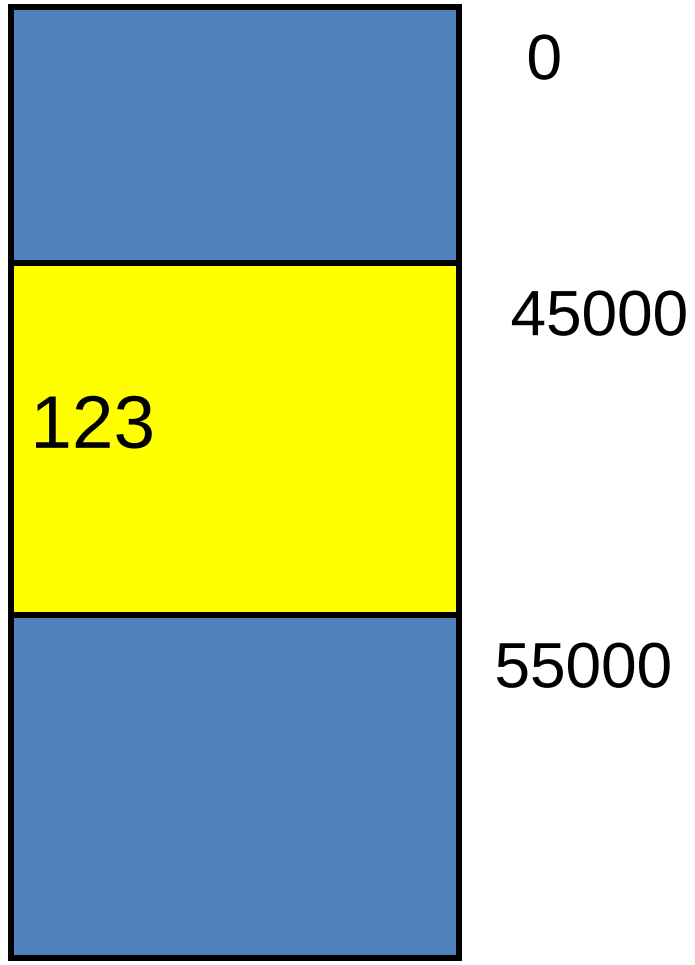
Limit Register

- The limit register contains the highest address of the program.
- The memory system checks each memory reference to make sure it is not greater than the value in the limit register.
- An addressing exception interrupt occurs if the address is too large.

Base Address with Limits

Base Address Register

45000



Program address 123 is located in RAM at the real physical address 45123

Base Address Translation

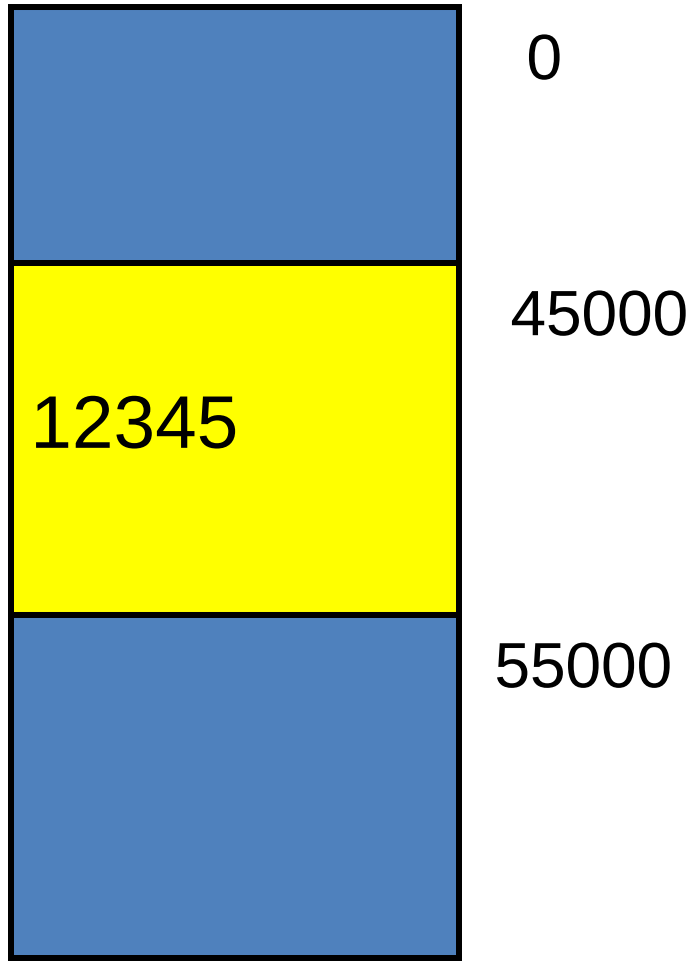
Base Address Register

45000

Limit Register

10000

Program address 12345 is too large



How can the base address system catch a negative address?

1. Use unsigned arithmetic
2. Have a positive and negative limit register
3. Reject addresses with the sign bit set
4. It never happens

Sharing

- Most programs in a Window's environment have code to draw the frame
- In a system with many programs, the frame drawing code is probably simultaneously being used by many programs
- It saves RAM if the same frame drawing instruction memory is not duplicated many times
- Each user needs their own data and stack memory

Memory Management Techniques

- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory
- Virtual Memory with segmentation

Fixed Partitioning

- Memory is divided into fixed size areas.
- Programs are loaded into these partitions.
- Often there were different sized partitions and partitions for different classes of jobs.
- Ancient operating systems such as OS/MFT or IBM DOS/VSE *(for mainframes, not the PC)* use fixed partitioning.

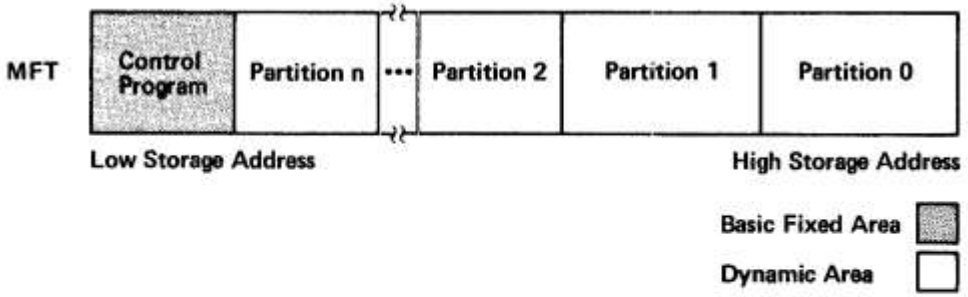
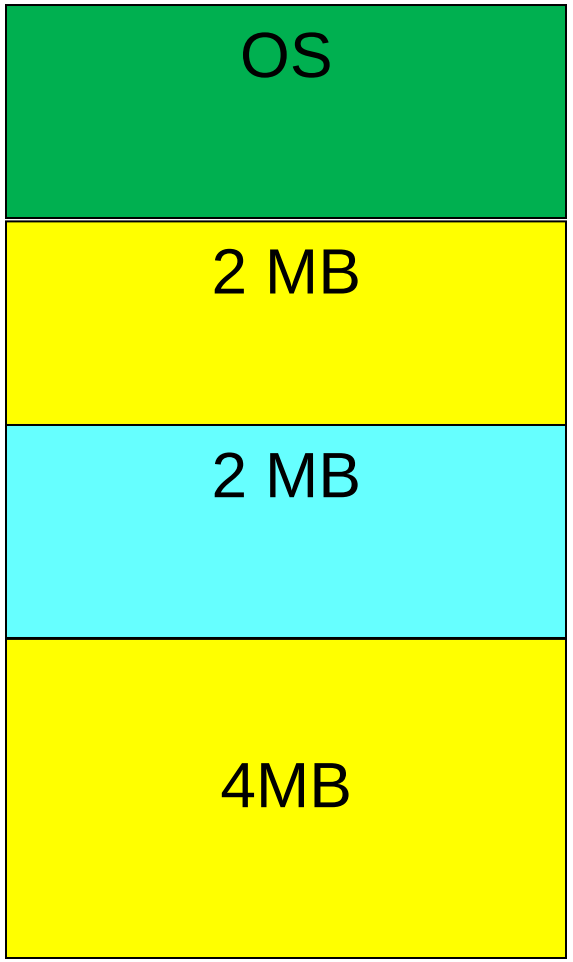


Figure 38. General Organization of Main Storage For the MFT Control Program Configuration

from IBM System/360 Operating System Introduction

Fixed Partitions

- Fixed partitions are easy to implement.
- A program fits in a partition or it doesn't.
- Internal fragmentation is high. No external fragmentation.
- Unable to run large programs when RAM is available.



Dynamic Partitioning

- Memory is allocated in varying sized blocks based on the size of the request.
- Memory is allocated when programs are loaded and released when the program terminates.
- Possibly high external fragmentation.

Function View of Memory Allocation

- **`alloc_mem(size);`**
  - Returns the address of an area of memory of the specified size.
  - May return an error status if no memory is available.
- **`free_mem(location, size);`**
  - Makes the previously allocated memory available again.
  - Adjacent free spaces need to be merged into one large free space.

Dynamic Partition Allocation Algorithms

- First Fit
- Best Fit
- Worst Fit
- Next Fit
- Last Fit
- Buddy system

On beyond OS

- These memory allocation algorithms have use in other areas. They can be used in any situation where a continuous resource is allocated in pieces.
- Other application areas are disk space allocation, heap allocation, equipment allocation in time, room reservation, etc.

First Fit Algorithm

- The OS keeps a list of the available free memory spaces sorted from low to high address.
- To find a location for a program, the OS searches the list from the beginning stopping when it finds an area equal to or greater than the needed size.
- If the space is bigger than needed, the left over space is put back on the free list.

Next Fit Algorithms

- This is a variation of the First Fit algorithm.
- Instead of starting the search at the beginning of the list each time as First Fit does, Next Fit saves the location of the last allocated space and start the next search from there.

Best Fit Algorithm

- The OS keeps a list of the available free memory spaces sorted from low to high address.
- To find a space of size X, the OS searches the entire list to find the hole that is the closest in size to X, but at least as big as X.

Worst Fit Algorithm

- The OS keeps a list of the available free memory spaces sorted from low to high address.
- The OS always allocates memory from the beginning of the largest available slot.

Last Fit Algorithm

- As memory is released, the free spaces are kept on a stack.
- Memory is allocated on a First Fit basis searching the stack from the most recently released to the earliest released.
- Takes advantage of cache. The most recently released memory is likely to be in cache.
- Used more often at the user level.

Buddy System

- Memory is always allocated in blocks that are a power of 2 in size.
- Separate lists are kept for free memory of sizes 2^i, 2^{i+1}, 2^{i+2}, …
- If a 2^k sized block is needed but no 2^k sized blocks are available, a 2^{k+1} block is split in two.
- The splitting process is repeated recursively.

Allocation Example

- Assume there is 1600 units of memory to allocate.
- Memory is allocated in 100 unit chunks.
- Programs W, X, Y and Z are initially loaded in memory as follows.

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

Allocation Example

time	program	size	action
1	A	200	allocate
2	B	300	allocate
4	C	100	allocate
3	Y	200	release
5	D	100	allocate
6	A	200	release
7	E	300	allocate
8	F	100	allocate

First Fit

- Allocate the memory requests in the previous table in the specified order.

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	D
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	D
200	C
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	E
500	E
400	E
300	D
200	C
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	E
500	E
400	E
300	D
200	C
100	
000	F

Example Allocation - Best Fit

- Allocate the same requests using the Best Fit algorithm.

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	A
600	A
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	A
600	A
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	A
600	A
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	A
600	A
500	
400	
300	
200	
100	
000	

1500	D
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	A
600	A
500	
400	
300	
200	
100	
000	

1500	D
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	
600	
500	
400	
300	
200	
100	
000	

1500	D
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	
600	
500	
400	
300	
200	E
100	E
000	E

1500	D
1400	W
1300	B
1200	B
1100	B
1000	Z
900	C
800	X
700	
600	
500	
400	
300	F
200	E
100	E
000	E

Example Allocation - Next Fit

- Allocate the same requests using the Next Fit algorithm.
- We need to keep track of the last allocation location.

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	
500	
400	
300	
200	
100	A
000	A

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	
500	
400	
300	
200	D
100	A
000	A

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	
500	
400	
300	
200	D
100	
000	

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	
500	E
400	E
300	E
200	D
100	
000	

1500	C
1400	W
1300	C
1200	C
1100	C
1000	Z
900	
800	X
700	
600	F
500	E
400	E
300	E
200	D
100	
000	

Example Allocation - Worst Fit

- Allocate the same memory using the Worst Fit algorithm

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	D
200	C
100	A
000	A

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	
500	
400	
300	D
200	C
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	E
500	E
400	E
300	D
200	C
100	
000	

1500	
1400	W
1300	B
1200	B
1100	B
1000	Z
900	
800	X
700	
600	E
500	E
400	E
300	D
200	C
100	
000	F

Example Allocation Buddy System

- Allocate the same memory requests using the Buddy System
- Note that the memory is divided by the powers of two.

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	
600	
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	A
600	A
500	Y
400	Y
300	
200	
100	
000	

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	
800	X
700	A
600	A
500	Y
400	Y
300	
200	B
100	B
000	B

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	C
800	X
700	A
600	A
500	Y
400	Y
300	
200	B
100	B
000	B

1500	
1400	W
1300	
1200	
1100	
1000	Z
900	C
800	X
700	A
600	A
500	
400	
300	
200	B
100	B
000	B

1500	D
1400	W
1300	
1200	
1100	
1000	Z
900	C
800	X
700	A
600	A
500	
400	
300	
200	B
100	B
000	B

1500	D
1400	W
1300	
1200	
1100	
1000	Z
900	C
800	X
700	
600	
500	
400	
300	
200	B
100	B
000	B

1500	D
1400	W
1300	
1200	
1100	
1000	Z
900	C
800	X
700	
600	E
500	E
400	E
300	
200	B
100	B
000	B

1500	D
1400	W
1300	
1200	
1100	F
1000	Z
900	C
800	X
700	
600	E
500	E
400	E
300	
200	B
100	B
000	B

Algorithm Commonality

Sort the list of free space by:

- Memory address *(First Fit)*
- Increasing size *(Best Fit)*
- Decreasing size *(Worst Fit)*
- Increasing time since last use *(Last Fit)*

The free list must be efficient for release as well as allocation.

Analysis of the Algorithms

- First Fit
  - Simple
  - Tends to put all of the programs on one end of memory.
  - May create lots of small holes in the front.
- Next Fit
  - May break up large holes

Analysis (cont.)

- Best Fit
  - Tends to leave lots of small holes.
  - Requires more searching
- Worst Fit
  - Not as bad as one might think
  - Tends to break up large holes
- Buddy System
  - Efficient allocation and release
  - Internal fragmentation if request size is not a power of 2

Microsoft Windows and Linux use

1. First fit
2. Next fit
3. Best fit
4. None of the above

Small Systems

- Small simple computers, such as embedded devices and wireless sensors, still use simple memory allocation.
- Handheld devices without mass storage use these memory allocation algorithms
- Most modern computers use virtual memory.